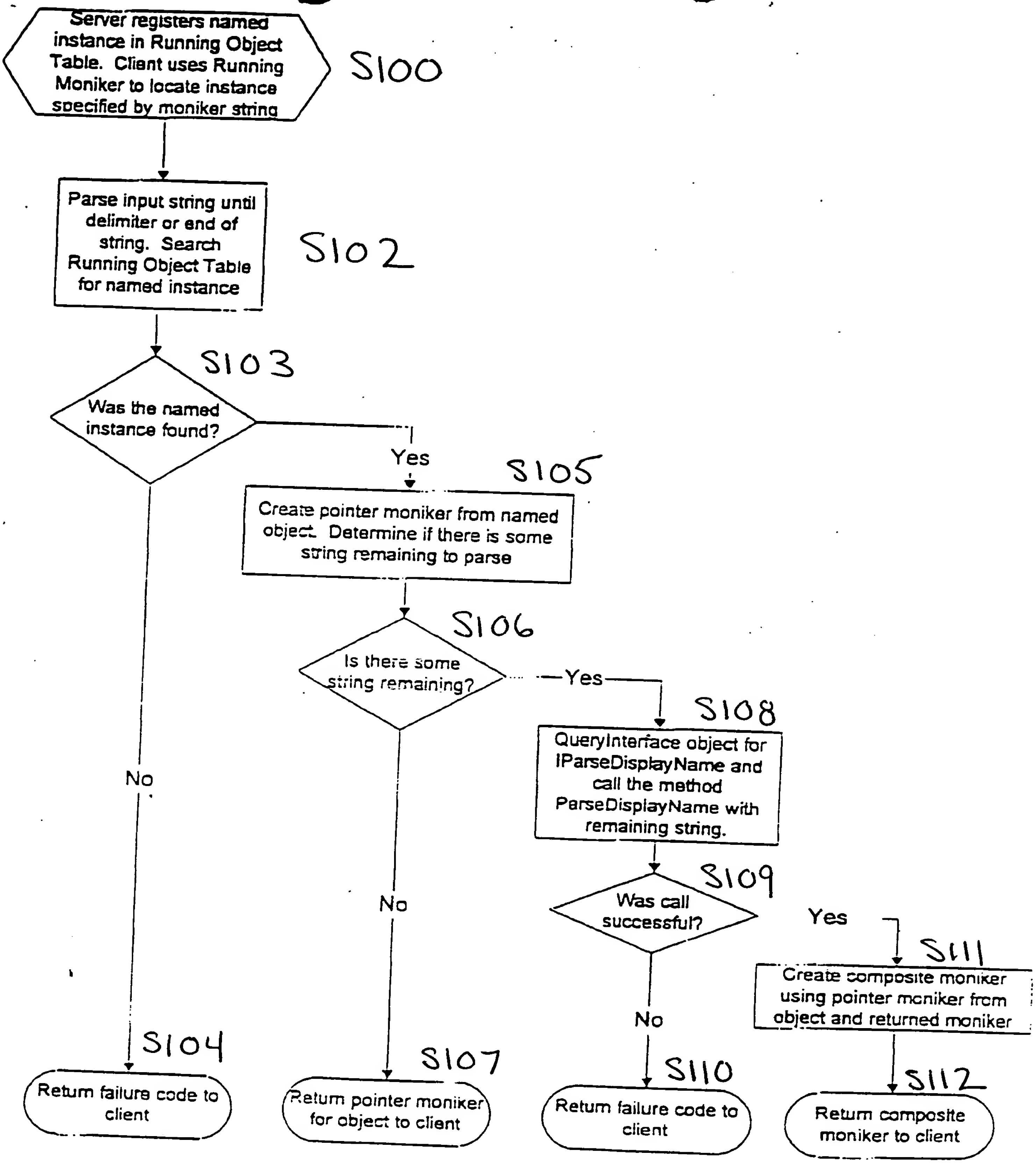
Server registers named instance in Running Object Table. Client uses Running Moniker to locate instance specified by moniker string
S100
Parse input string until delimiter or end of string. Search Running Object Table for named instance
S102
S103
Was the named instance found?
Yes
S105
Create pointer moniker from named object. Determine if there is some string remaining to parse
S106
Is there some string remaining?
Yes
S108
QueryInterface object for IParseDisplayName and call the method ParseDisplayName with remaining string.
S109
Was call successful?
Yes
S111
Create composite moniker using pointer moniker from object and returned moniker
No
No
No
S104
S107
S110
S112
Return failure code to client
Return pointer moniker for object to client
Return failure code to client
Return composite moniker to client

Fig. 1

```
// Running.idl : IDL source for Running.dll
//

// This file will be processed by the MIDL tool to
// produce the type library (Running.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";

        [
                object,
                uuid(E7531917-289D-11D2-869F-080009DC2552),
                dual,
                helpstring("IRunning Interface"),
                pointer_default(unique)
        ]
        interface IRunning : IDispatch
        {
                [id(1), helpstring("method RegisterInstanceName")] HRESULT RegisterInstanceName(BSTR bstr
ItemName, IUnknown * pUnk, long * lCookie);
                [id(2), helpstring("method UnregisterInstanceName")] HRESULT UnregisterInstanceName(long
lCookie);
        };
[
        uuid(E7531908-289D-11D2-869F-080009DC2552),
        version(1.0),
        helpstring("Running 1.0 Type Library")
]
library RUNNINGLib
{
        importlib("stdole32.tlb");
        importlib("stdole2.tlb");

        [
                uuid(E7531918-289D-11D2-869F-080009DC2552),
                helpstring("Running Class")
        ]
        coclass Running
        {
                [default] interface IRunning;
                interface IParseDisplayName;
        };
};
```

Fig. 2A

```
                                        Running.rgs
HKCR
{
        Running.1 = s 'Running Class'
        {
                CLSID = s '{E7531918-289D-11D2-869F-080009DC2552}'
        }
        Running = s 'Running Class'
        {
                CLSID = s '{E7531918-289D-11D2-869F-080009DC2552}'
        }
        NoRemove CLSID
        {
                ForceRemove {E7531918-289D-11D2-869F-080009DC2552} = s 'Running Class'
                {
                        val AppID = s '{E7531918-289D-11D2-869F-080009DC2552}'
                        ProgID = s 'Running.1'
                        VersionIndependentProgID = s 'Running'
                        ForceRemove 'Programmable'
                        InprocServer32 = s '%MODULE%'
                        {
                                val ThreadingModel = s 'Apartment'
                        }
                }
        }
        NoRemove AppID
        {
                NoRemove {E7531918-289D-11D2-869F-080009DC2552} = s 'Running Class'
                {
                    val DllSurrogate = s ''
                }
        }
}
```

Fig.2B

CRunning.h

```
// CRunning.h : Declaration of the CRunning

#ifndef __RUNNING_H_
#define, __RUNNING_H_

#include "resource.h"       // main symbols

//////////////////////////////////////////////////////////////////////////////
// CRunning
class ATL_NO_VTABLE CRunning :
        public CComObjectRootEx<CComSingleThreadModel>,
        public CComCoClass<CRunning, &CLSID_Running>,
        public IDispatchImpl<IRunning, &IID_IRunning, &LIBID_RUNNINGLib>,
        public IParseDisplayName
{
public:
        CRunning()
        {
                ATLTRACE(_T("CRunning() constructor called\n"));
        }

        virtual ~CRunning()
        {
                ATLTRACE(_T("CRunning() destructor called\n"));
        }

DECLARE_REGISTRY_RESOURCEID(IDR_RUNNING)

BEGIN_COM_MAP(CRunning)
        COM_INTERFACE_ENTRY(IRunning)
        COM_INTERFACE_ENTRY(IDispatch)
        COM_INTERFACE_ENTRY(IParseDisplayName)
END_COM_MAP()

//////////////////////////////////////////////////////////////////
// IParseDisplayName method

        STDMETHODIMP ParseDisplayName(IBindCtx *pbc
                                                                                ,LPOLESTR pszDisplayName
                                                                                ,ULONG *pchEaten
                                                                                ,IMoniker **ppmkOut
                                                                                );

protected:
        const wchar_t* ProgID() { return L"Running"; }
    const wchar_t* VersionIndependentProgID() {  return L"Running.1"; }

// IRunning
public:
        STDMETHOD(UnregisterInstanceName)(long lCookie);
        STDMETHOD(RegisterInstanceName)(BSTR bstrItemName, IUnknown * pUnk, long * lCookie);
};

#endif //__RUNNING_H_
```

Fig.2C

CRunning.cpp

```
// CRunning.cpp : Implementation of CRunning
#include "stdafx.h"
#include "Running.h"
#include "CRunning.h"


#define BAD_POINTER_RETURN(p) if( !p ) return E_POINTER
#define BAD_POINTER_RETURN_OR_ZERO(p) if( !p ) return E_POINTER; else *p = 0
#define SIZE_OF_STRING(p) !p ? 0 : ((wcslen(p) * sizeof(wchar_t)) + sizeof(wchar_t))

#define OLE_MAXNAMESIZE 256


/////////////////////////////////////////////////////////////////////////////
// CRunning


STDMETHODIMP CRunning::RegisterInstanceName(BSTR bstrItemName, IUnknown * pUnk, long * lCookie)
{
        AFX_MANAGE_STATE(AfxGetStaticModuleState())

        // TODO: Add your implementation code here
        ATLTRACE(_T("CRunning::RegisterInstanceName called\n"));

        HRESULT hr = E_FAIL;

        LPRUNNINGOBJECTTABLE  prot = NULL;

        hr = GetRunningObjectTable(0,&prot);
        if(SUCCEEDED(hr))
        {
                LPMONIKER       ppmk = NULL;

                hr = CreateItemMoniker(NULL,bstrItemName,&ppmk);

                if(SUCCEEDED(hr))
                {
                        hr = prot->Register(0
                                                                ,pUnk
                                                                ,ppmk
                                                                ,(unsigned long *)lCookie
                                                                );

                        if(SUCCEEDED(hr))
                        {
                                TRACE(_T("CRunning::RegisterInstanceName register succeeded cookie is %x\
n"),(unsigned long*)*lCookie);
                        }
                        else
                        {
                                TRACE(_T("CRunning::RegisterInstanceName register failed %x \n"),hr);
                        }
                        ppmk->Release();
                }
                else
                {
                        TRACE(_T("CRunning::RegisterInstanceName CreateItemMoniker failed %x \n"),hr);
                }
                prot->Release();
        }
        else
        {
                TRACE(_T("CRunning::RegisterInstanceName get ROT failed %x \n"),hr);
        }
        return hr;
}

STDMETHODIMP CRunning::UnregisterInstanceName(long lCookie)
{
        AFX_MANAGE_STATE(AfxGetStaticModuleState())

        // TODO: Add your implementation code here
        HRESULT hr = E_FAIL;
```

Fig. 2D

```
        if(lCookie)
        {
                LPRUNNINGOBJECTTABLE    prot = NULL;

                hr = GetRunningObjectTable(0,&prot);

                if(SUCCEEDED(hr))
                {
                        hr = prot->Revoke((unsigned long)lCookie);
                        if(SUCCEEDED(hr))
                        {
                                TRACE(_T("CRunning::UnregisterInstanceName worked for cookie %x \n"),(uns
igned long)lCookie);
                        }
                        else
                        {
                                ATLTRACE(_T("CRunning::UnregisterInstanceName Revoke failed\n"));
                        }
                        prot->Release();
                }
                else
                {
                        ATLTRACE(_T("CRunning::UnregisterInstanceName GetROT failed\n"));
                }
        }
        return hr;
}


STDMETHODIMP CRunning::ParseDisplayName(
    IBindCtx*    pbc,
    LPOLESTR     pwszDisplayName,
    ULONG*       pchEaten,
    IMoniker**   ppmkOut)
{
        AFX_MANAGE_STATE(AfxGetStaticModuleState())

        ATLTRACE(_T("CRunning::ParseDisplayName() with %S\n"),pwszDisplayName);

        BAD_POINTER_RETURN_OR_ZERO(ppmkOut);
    BAD_POINTER_RETURN_OR_ZERO(pchEaten);
    BAD_POINTER_RETURN(pbc);
    BAD_POINTER_RETURN(pwszDisplayName);
    BAD_POINTER_RETURN(pchEaten);

        ATLTRACE(_T("CRunning::ParseDisplayName() pointers OK!\n"));

        HRESULT hr = E_FAIL;

        // set to max for now
        // need to change to fit MkParseEx
        if(*pwszDisplayName == L'@')
                * pchEaten = wcslen(L"@Running");
        else
                * pchEaten = wcslen(L"Running");

        // as far as i have been able to find oud
        // MkParse will pass the @, WRONG!!
        // oh no!!! MkParseEx doesn't pass the @!!
        // we've got to fix this,  so let's look for ":"

        wchar_t * pwszInstance = wcschr(pwszDisplayName,L':');

        // do we have an instance ?

        if(pwszInstance)
        {
                ATLTRACE(_T("CRunning::ParseDisplayName() instance name %S\n"),pwszInstance);

                WCHAR           szItemName[OLE_MAXNAMESIZE];
                LPWSTR          lpszDest        = szItemName;
                LPWSTR          lpszSrc         = pwszInstance;
                int                     cEaten = 0;
```

Fig. 2E

CRunning.cpp

```
        // eat delimiter characters until next token
        while (*lpszSrc != L'\0' && (*lpszSrc == L'\\' || *lpszSrc == L'/' ||
                *lpszSrc == L':' || *lpszSrc == L'!' || *lpszSrc == L'['))
        {
                ++lpszSrc;
                ++cEaten;
        }

        // parse next token in szItemName
        while (*lpszSrc != L'\0' && *lpszSrc != L'\\' && *lpszSrc != L'/' &&
                        *lpszSrc != L':' && *lpszSrc != L'!' && *lpszSrc != L'[' &&
                        cEaten < OLE_MAXNAMESIZE-1)
        {
                *lpszDest++ = *lpszSrc++;
                ++cEaten;
        }
        *pchEaten += cEaten;
        *lpszDest = 0;

        // find the running object
        LPRUNNINGOBJECTTABLE    prot    = NULL;
        LPENUMMONIKER                   penum = NULL;
        LPMONIKER                               ppmk    = NULL;

        hr = CreateItemMoniker(NULL,szItemName,&ppmk);

        ATLTRACE(_T("CRunning::ParseDisplayName() CreateItemMoniker %x\n"),hr);

        if(SUCCEEDED(hr))
        {
                // look in the running object table to find the gizmo

                // since we are a moniker provider we can't use
                // the bind context to get the ROT
                hr = GetRunningObjectTable(0, &prot);
                ATLTRACE(_T("CRunning::ParseDisplayName() GetRunningObjectTable %x\n"),hr);
                if(SUCCEEDED(hr))
                {
                        hr = prot->EnumRunning(&penum);
                        ATLTRACE(_T("CRunning::ParseDisplayName() EnumRunning %x\n"),hr);
                        if(SUCCEEDED(hr))
                        {
                                IMoniker        *       ppmkTest        = NULL;
                                IMoniker        *       ppmkResult      = NULL;
                                IUnknown        *       pUnk            = NULL;
                                BOOL                    bFound          = FALSE;

                                while((penum->Next(1,&ppmkTest,NULL) == S_OK)&&(!bFound))
                                {
                                        hr = ppmk->IsEqual(ppmkTest);
                                        if(hr == S_OK) // not SUCCEEDED!!
                                        {
t\n"));                                         TRACE(_T("CRunning::ParseDisplayName() we found i

                                                bFound = TRUE;
                                                hr = prot->GetObject(ppmkTest,&pUnk);
                                                if(hr == S_OK) // not SUCCEEDED!!
                                                {
 got it\n"));                                           TRACE(_T("CRunning::ParseDisplayName() we

t);                                                     hr = CreatePointerMoniker(pUnk,&ppmkResul

                                                        if(SUCCEEDED(hr))
                                                        {
ame() created pointer moniker\n"));                             TRACE(_T("CRunning::ParseDisplayN

= NULL;                                                         IParseDisplayName       * pParse

 pItemMoniker = NULL;                                           IMoniker                *

 ucEaten;                                                       ULONG
```

Fig. 2F

CRunning.cpp

```
s correct
s to with it



'(IID_IParseDisplayName,(void **)&pParse);


eDisplayName(pbc,lpszSrc,&ucEaten,&pItemMoniker);


 += ucEaten;
Result->ComposeWith(pItemMoniker,FALSE,ppmkOut);
DED(hr))

RACE(_T("CRunning::ParseDisplayName() It worked!!!\n"));
/ we can release the constituant elements
/ of the composite
pmkResult->Release();

succeed or fail we can release the
oniker
ker->Release();

;
```

```
// we'll give him the part that i
// and he can do whatever he want
*ppmkOut = ppmkResult;
// is there any string to parse?
if(*lpszSrc != L'\0')
{
        hr = pUnk->QueryInterface
        if(SUCCEEDED(hr))
        {
                hr = pParse->Pars
                if(SUCCEEDED(hr))
                {
                        *pchEaten
                        hr = ppmk
                        if(SUCCEE
                        {
                                T
                                /
                                /
                                p
                        }
                        // if we
                        // item m
                        pItemMoni
                }
                pParse->Release()
        }
}
                                    }
                               }
                          }
                          ppmkTest->Release();
                     }

                     if(!bFound)
                     {
                          hr = E_FAIL;
                     }
                     penum->Release();
                }
                prot->Release();
           }
           ppmk->Release();
      }
   }
   return hr;
}
```

Fig. 2G

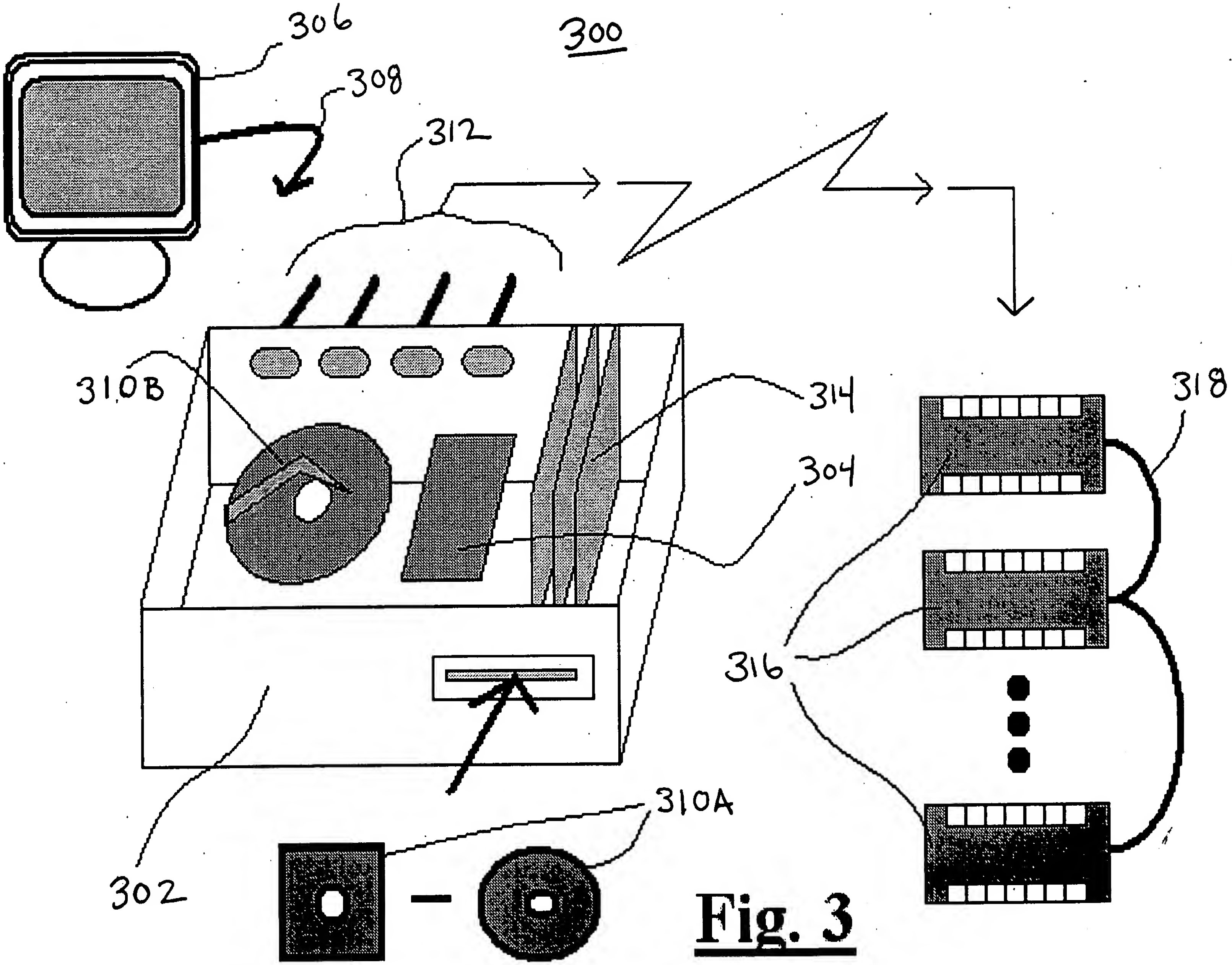

Fig. 3

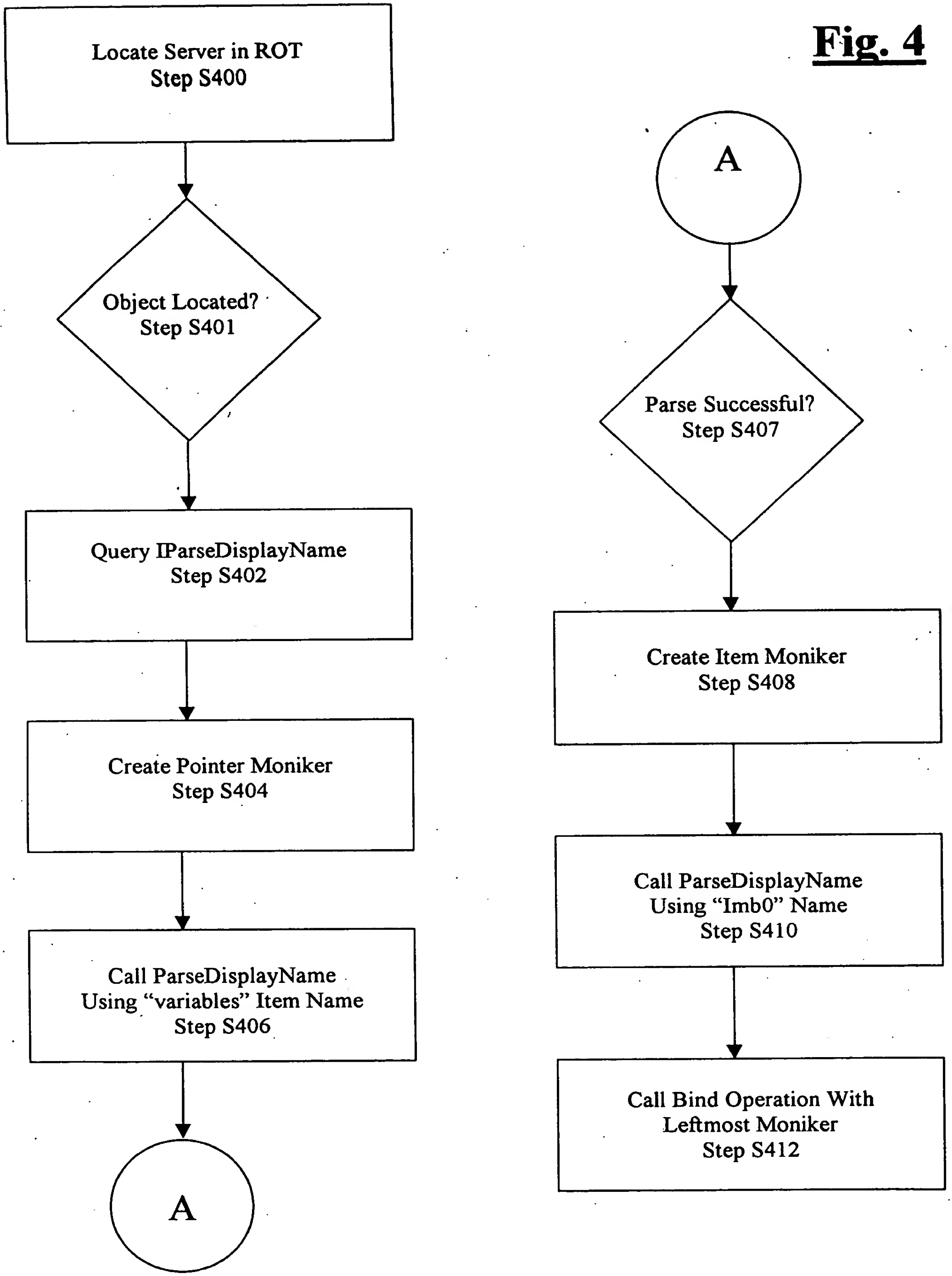
Locate Server in ROT
Step S400
Object Located?
Step S401
Query IParseDisplayName
Step S402
Create Pointer Moniker
Step S404
Call ParseDisplayName
Using “variables” Item Name
Step S406
A
A
Parse Successful?
Step S407
Create Item Moniker
Step S408
Call ParseDisplayName
Using “Imb0” Name
Step S410
Call Bind Operation With
Leftmost Moniker
Step S412

Fig. 4